Aplicación de Qt y Scalable Vector Graphics al diseño de controles visuales para sistemas industriales

Índice

1. Objetivo del proyecto.

El objetivo perseguido en la realización de este proyecto, es el de conseguir iconos industriales de código abierto y con opción de multiplataforma.

Para la realización del proyecto utilizamos el lenguaje de programación C/C++, dado que es un lenguaje conocido y utilizado en múltiples plataformas, siendo un lenguaje muy potente.

En el aspecto de obtener opción multiplataforma, nos referimos a la posibilidad de poder utilizar los iconos industriales en distintos dispositivos con diferentes sistemas operativos. Estos sistemas pueden ser:

- Linux
- Windows
- Symbian

De este modo podemos obtener aplicaciones para controlar sistemas industriales a través de diversos dispositivos, sin tener que limitarnos a una opción únicamente.

2. Solución propuesta: SVG + QtSVG.

Por la necesidad de utilizar código abierto y sin necesidad de pagar licencias de ningún tipo, la solución que proponemos es la utilización de imágenes SVG, configurando movimientos y otras propiedades para los iconos industriales a través del programa QtSVG, el cual es libre para la utilización en código abierto y de este modo olvidarnos de la necesidad de diferentes licencias para su legal utilización.

3. ¿Qué es SVG? Ejemplos.

Scalable Vector Graphics (**SVG**) es una especificación para describir gráficos vectoriales bidimensionales, tanto estáticos como animados (estos últimos con ayuda de SMIL), en formato XML.

El SVG permite tres tipos de objetos gráficos:

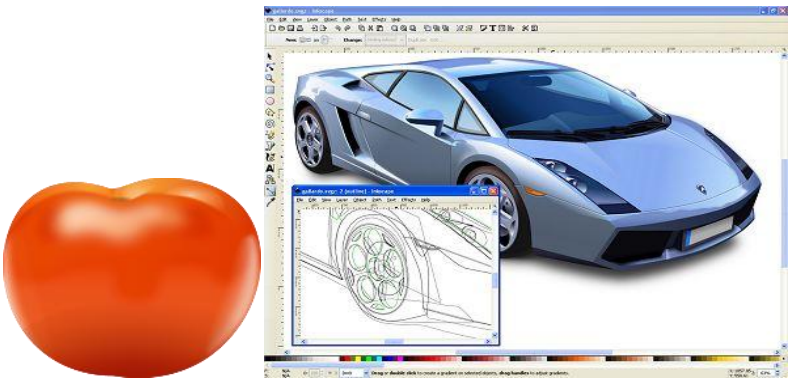
- Formas gráficas de vectores (p.e. caminos consistentes en rectas y curvas, y áreas limitadas por ellos)
- Imágenes de mapa de bits /digitales
- Texto

Los objetos gráficos pueden ser agrupados, transformados y compuestos en objetos previamente renderizados, y pueden recibir un estilo común. El texto puede estar en cualquier espacio de nombres XML admitido por la aplicación, lo que mejora la posibilidad de búsqueda y la accesibilidad de los gráficos SVG. El juego de características incluye las transformaciones anidadas, los *clipping paths*, las máscaras alfa, los filtros de efectos, las plantillas de objetos y la extensibilidad.

El dibujado de los SVG puede ser dinámico e interactivo. El Document Object Model (DOM) para SVG, que incluye el DOM XML completo, permite animaciones de gráficos vectoriales sencillas y eficientes, mediante ECMAScript o SMIL. Un juego amplio de manejadores de eventos, como "onMouseOver" y "onClick", pueden ser asignados a cualquier objeto SVG. Debido a su compatibilidad y relación con otras normas Web, características como el scripting pueden ser aplicadas a elementos SVG y a otros elementos XML desde distintos espacios de nombre XML simultáneamente dentro de la misma página web. Un ejemplo extremo de esto es un juego completo de tetris realizado como un objeto SVG.

Si el espacio de almacenamiento es un problema, las imágenes SVG pueden salvarse comprimidas con gzip, en cuyo caso pasan a ser imágenes SVGZ. Debido a la verbosidad del XML, este tiende a comprimirse muy bien, y estos ficheros pueden ser mucho más pequeños. Aun así, a menudo el fichero vectorizado original (SVG) es más pequeño que la versión de mapa de bits.

A pesar de ser un lenguaje vectorial, SVG permite crear imágenes complejas.



El uso de SVG hoy en día es muy común y a continuación se detallan navegadores que trabajan con este sistema de imagen.

- **Firefox**: implementa SVG en forma nativa desde su versión 1.5. A través del tiempo fue mejorando el cumplimiento del estándar, pero con alto consumo de procesador. En la nueva versión 3.5 de Firefox se puede comprobar que el render SVG se ha modificado y mejorado.
- **Opera**: al igual que Firefox también implementa SVG en forma nativa, pero con poco consumo de procesador. La versión 9.5 Beta, incorpora la posibilidad de llamar en forma externa a una imagen en formato svg. Usando <image> o <use>.
- **MSIE**: No implementa directamente SVG, por lo que se debe conseguir una extensión de la firma Adobe. Este módulo externo no permite mezclar SVG con HTML (XHTML). Microsoft no ve con futuro a este estándar por lo que apostó al VML. El 5 de enero de 2010, Microsoft publicó que pasarían a formar parte del grupo de trabajo SVG en la W3C.
- **Safari**: Su versión 3.1 (para computadores con sistema operativo Windows o Mac OS X) implementa SVG tanto para imágenes como para texto avanzado.
- **Chrome**: Desde su primera versión implementa SVG de forma nativa (ya que utiliza WebKit).

4. Editores de SVG

A la hora de realizar imágenes de formato SVG tenemos diferentes opciones, y como ocurre habitualmente, podemos optar por programas libres de licencias y condiciones (programas gratuitos) o programas de los cuales tenemos que obtener una licencia para su utilización (programas de pago).

Dentro de los programas gratuitos la opción utilizada en este proyecto y recomendada es InkScape, es un programa perteneciente al entrono de Linux, pero también tenemos la versión para Windows, este programa es fácilmente utilizable y posee una gran variedad de comandos para poder editar imágenes con múltiples características. También existen otros programas gratuitos para la edición de imágenes con formato svg, pero a continuación vamos a hablar sobre InkScape.

InkScape es un editor de gráficos vectoriales de código abierto, con capacidades similares a Illustrator, Freehand, CorelDraw o Xara X, usando el estándar de la W3C: el formato de archivo Scalable Vector Graphics (SVG). Las características soportadas incluyen: formas, trazos, texto, marcadores, clones, mezclas de canales alfa, transformaciones, gradientes, patrones y agrupamientos. Inkscape también soporta meta-datos Creative Commons, edición de nodos, capas, operaciones complejas con trazos, vectorización de archivos gráficos, texto en trazos, alineación de textos, edición de XML directo y mucho más. Puede importar formatos como Postscript, EPS, JPEG, PNG, y TIFF y exporta PNG así como muchos formatos basados en vectores.

El objetivo principal de Inkscape es crear una herramienta de dibujo potente y cómoda, totalmente compatible con los estándares XML, SVG y CSS. También queremos mantener una próspera comunidad de usuarios y desarrolladores usando un sistema de desarrollo abierto y orientado a las comunidades, y estando seguros de que Inkscape sea fácil de aprender, de usar y de mejorar.

Dentro de los programas de pago también disponemos de distintas alternativas, aunque la más conocida es CorelDraw. En este caso no vamos a comentar nada respecto al programa porque la realización de todo el proyecto ha sido editada con InkScape.

5. QtSVG.

QtSVG es un modulo perteneciente a Qt. Al añadir este modulo en la creación de un proyecto, como hemos de realizar, conseguimos dotar a nuestro proyecto con la posibilidad de manejar imágenes de formato svg.

6. Solución propuesta.

Para conseguir distintos iconos industriales (widget's), utilizaremos la plataforma de dibujo InkScape y para la programación de dichas imágenes utilizaremos el programa Qt, incluyendo el modulo QtSVG.

Hemos escogido esta solución ya que buscamos la realización de widget's en código abierto, por este motivo hemos pensado que las aplicaciones indicadas anteriormente son las más adecuadas para conseguir nuestro objetivo, ya que Qt nos da la posibilidad de programar para distintas plataformas, que es uno de los objetivos que perseguimos.

Para comenzar la realización del proyecto, y una vez disponemos de todos los programas, obtenemos los archivos necesarios para otorgar movimiento a nuestros widget's. Estos archivos son comunes para los widget's del mismo tipo, de este modo se facilita la creación de una gran cantidad de widget's sin tener que editar largos archivos, solo variando las imágenes que queremos de los nuevos widget's.

7. Guía para construir tu propio widget con SVG.

Ahora vamos a editar una guía rápida a seguir para construir nuestro propio widget de forma rápida y eficaz.

Primero detallar que cuando realicemos la imagen de nuestro widget, tiene que tener unas características

Para conseguir los widget's comenzaremos indicando la utilización de los archivos necesarios que otorgan diferentes clases de movimiento a las imágenes para los widget's que pretendemos realizar.

Podemos clasificar estos widget's en tres grandes grupos:

- Manómetros, etc. (Primer tipo).
- Barras, indicadores de nivel (Segundo tipo).
- Válvulas, diferentes estados de posición (Tercer tipo).

✓ Antes de empezar con la programación y edición del proyecto hay que saber que para la elaboración de las imágenes y su perfecta visualización hay que respetar unas series de normas.

- Una vez dibujada la imagen, ajustaremos el tamaño de la hoja al de la imagen.
- Las distintas imágenes para el mismo widget, tienen que estar en distintos archivos, pero situadas en su correcta ubicación, como si estuvieran en el mismo archivo.
- Las imágenes de *Segundo tipo,* la imagen correspondiente al nivel que va a aumentar y disminuir, tiene que estar editada en su máximo tamaño.

Ahora comenzaremos con la programación.

Comenzaremos con la creación de un *Proyecto Qt4 con GUI*, una vez le asignemos un nombre a nuestro proyecto (en este caso widget1) y una ubicación, nos aparecerá una lista con distintos módulos para agregar a nuestro proyecto, seleccionaremos *QtSVG*, continuaremos y finalizaremos la creación del proyecto. Se nos habrá creado un proyecto como el mostrado en la Figura1.

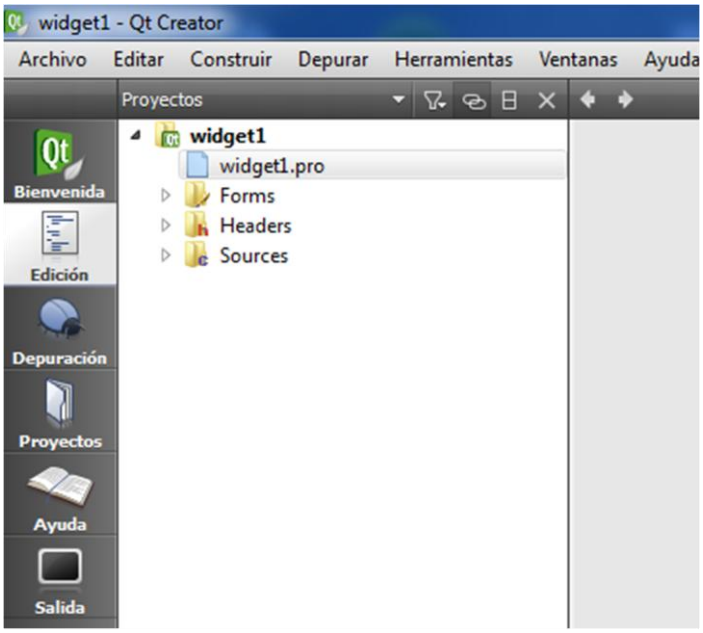


Figura 1

Ahora tenemos que añadir los archivos que dotaran de movimiento a los distintos Widget's y un archivo llamado *common* el cual hace posible la carga de imágenes. Para incluir estos archivos, tendremos que copiar los directorios (*common, multislider, scrollwheel y svgdialgauge*) dentro del directorio de nuestro proyecto y nos quedara como indica la Figura 2.

Figura 2

Para incluirlos en nuestro proyecto, deberemos escribir los "*include*" dentro del archivo widget1.pro:

```
include(./common/common.pri)
include(./multislider/multislider.pri)
include(./svgdialgauge/svgdialgauge.pri)
include(./scrollwheel/scrollwheel.pri)
```

una vez guardado quedara como muestra la Figura 3.

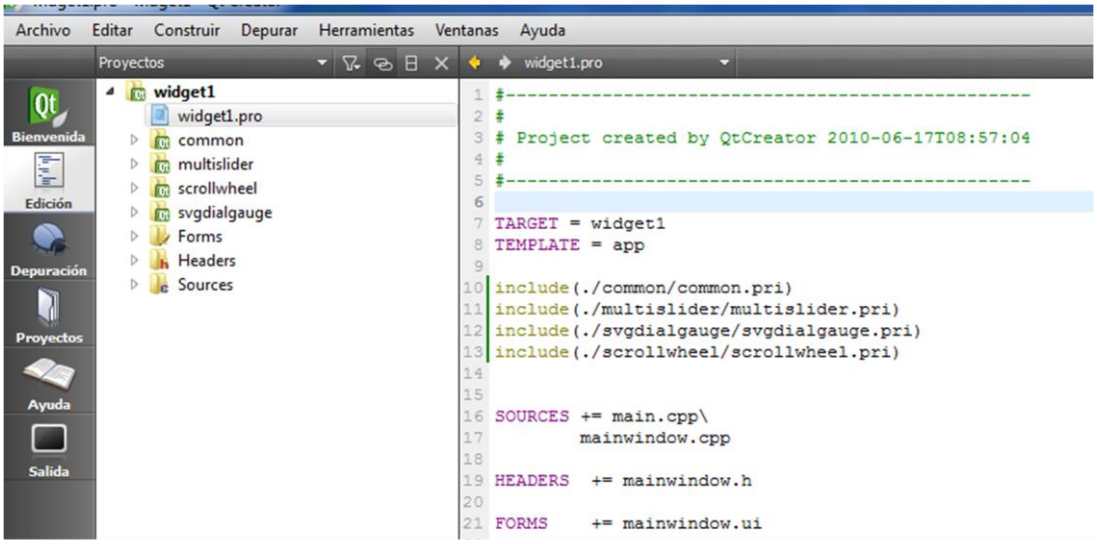


Figura 3

El siguiente paso a realizar será incluir las imágenes .svg dentro del proyecto. En primer lugar crearemos dentro de la carpeta de cada tipo de widget's (*multislider, scrollwheel y svgdialgauge*) una o varias carpetas para introducir las imágenes de cada widget. Las imágenes se introducirán dentro del tipo que correspondan. Por ejemplo, si pretendemos elaborar un widget que represente un depósito, las imágenes correspondientes a este widget se introducirán dentro de una carpeta creada dentro de *multislider*.

Las imágenes tienen que tener un nombre específico, a continuación se detalla el nombre que hay que asignar a las imágenes dependiendo que tipo de widget sea.

- Primer tipo: La imagen correspondiente al fondo, deberá tener el nombre "*background*" y el indicador (aguja) tendrá el nombre "*needle*".
- Segundo tipo: La imagen que quedaría fija (en caso de una deposito, seria la imagen del depósito) tendrá el nombre "*valuebar*" y la imagen que variara de tamaño (en caso del depósito, el liquido que contiene) se llamara "*valuebar_filled*".
- Tercer tipo: La imagen que representa el primer estado tendrá el nombre de "*Wheel_1*" la del segundo estado tendrá el nombre de "*Wheel_2*", este ejemplo se ha realizado con dos estados, pero se podrían introducir más.

✓ Por supuesto se pueden cambiar los nombres correspondientes a las imágenes pero esto implicaría editar los archivos *.cpp* en los cuales se indica el nombre de las imágenes a añadir.

Una vez introducidas todas las imágenes, en las carpetas correspondientes y con sus correspondientes nombres, lo que debemos hacer es incluirlas en nuestro proyecto, para esto crearemos dentro del correspondiente aparatado, en caso de un deposito, dentro de multislider un *Archivo de recursos Qt (.qrc)*. Luego abriremos este archivos y pulsaremos *Agregar Prefijo*, aquí escribiremos "*/multislider"* (hay que tener en cuenta que esto es para la creación de un deposito, en caso de la creación de un widget de tipo manómetro, realizaremos lo mismo pero en la carpeta *svgdialgauge*), a continuación pulsaremos *Agregar Archivos* y seleccionaríamos las imágenes. Una vez agregada las imágenes quedara como muestra la Figura 4.

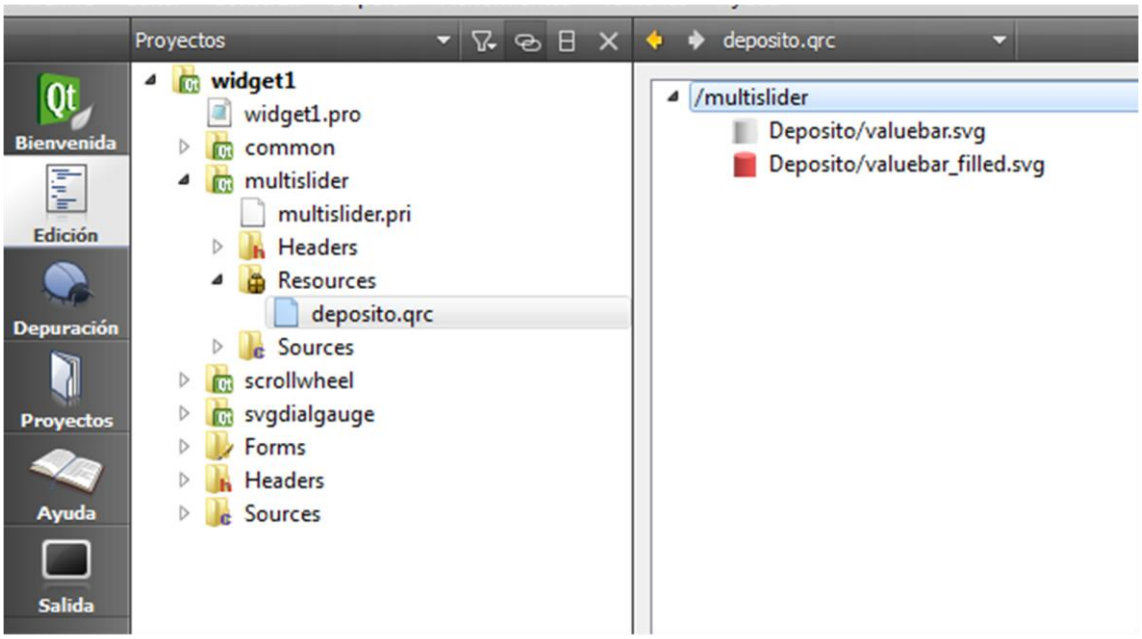


Figura 4

A continuación introduciremos en la cabecera de los archivos *mainwindow.h, mainwindow.cpp* y *main.cpp* los *include* para que todo funcione correctamente.

```
#include <QtSvgDialGauge>
#include <QtMultiSlider>
#include <QtScrollWheel>
```

Ahora comenzamos la edición de los distintos widget. En primer lugar dentro del archivo *mainwindow.h* crearemos unos vectores con los tipos de widget's que utilizaremos.

```
QtSvgDialGauge *gauge;
QtMultiSlider *multiSlider;
QtScrollWheel *scrollWheel;
```

Esto lo incluiremos en el espacio "*prívate*" como se observa en la Figura 5.

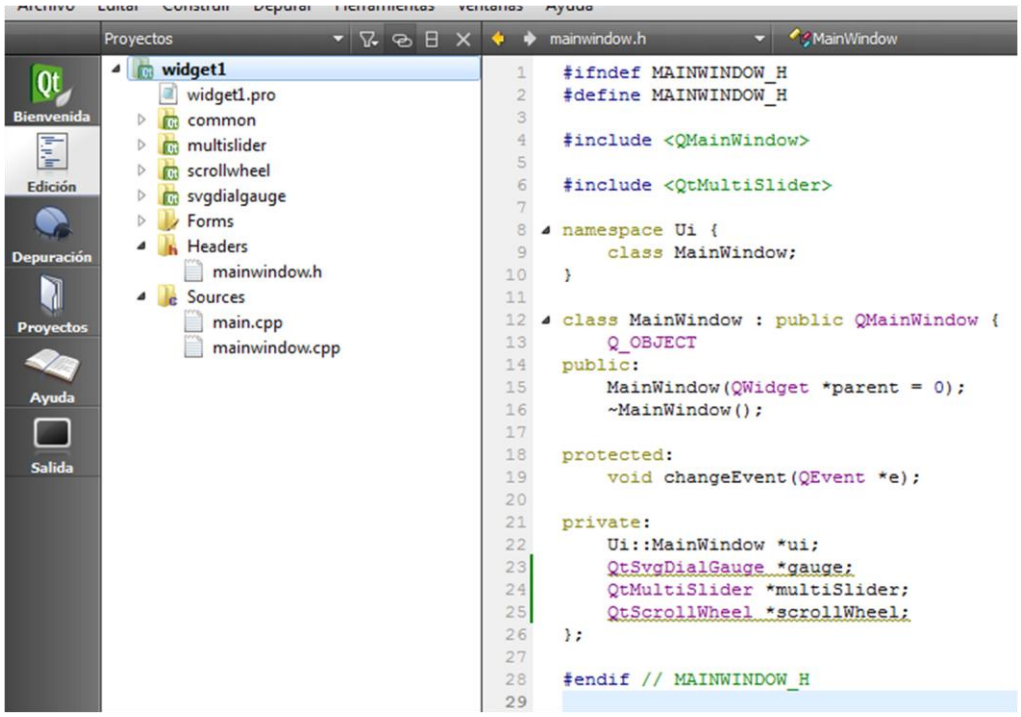


Figura 5

De este modo hemos creado 3 vectores, para la creación de tres widget's uno de cada tipo.

A continuación nos disponemos a editar el archivo *mainwindow.cpp*, en este archivo editaremos las propiedades de cada uno de los widget's. Aquí vamos a explicar la creación de tres widget's, un deposito, un tacómetro y una válvula.

Para la edición del depósito el código que deberemos editar es el siguiente:

```
//Deposito
multiSlider = new QtMultiSlider(this);
multiSlider->setSkin("Deposito");        //Nombre carpeta imagenes
multiSlider->setValue(0);                //Valor inicial del "liquido"
```

Para la edición de un tacómetro introduciremos lo siguiente:

```
// Tacometro
gauge = new QtSvgDialGauge(this);
gauge->setSkin("Tacometro");  //Nombre de la carpeta con las imagenes
gauge->setNeedleOrigin(0.486, 0.466);
gauge->setMinimum(0);
gauge->setMaximum(360);
gauge->setStartAngle(-130);              //Angulo Minimo
gauge->setEndAngle(133);                 //Angulo Maximo
gauge->setValue(0);                      //Valor inicial del que marca
gauge->setMaximumSize(200, 200);
```

Por último, para editar una válvula con dos estado introduciremos el siguiente código:

```
//Valvula
 scrollWheel = new QtScrollWheel(this);
 scrollWheel->setSkin("Valvula");          //Nombre carpeta imagenes
```

Llegados a este punto ya tenemos los widget's editados, lo último que falta es darles una ubicación dentro del archivo *mainwindow.ui* y asignarles un dispositivo que les otorgué el movimiento. En la Figura 6, podemos ver la disposición y los elementos que he colocado para controlar cada widget.

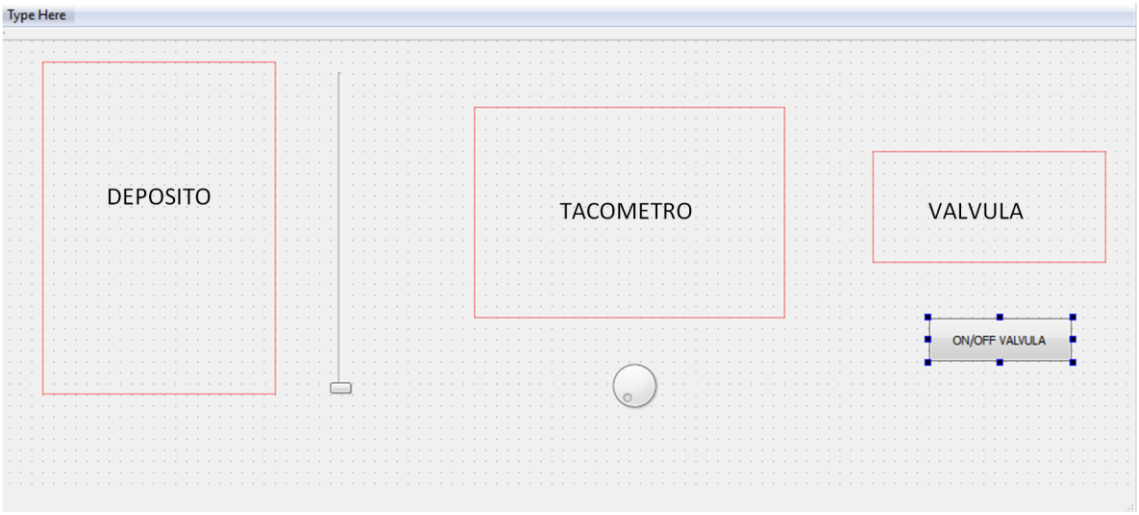


Figura 6

Para finalizar la programación, en el archivo *mainwindow.cpp* daremos la ubicación a cada widget en su *layout* y vincularemos los dispositivos de control con los widget's.

Para indicar la ubicación de cada widget, escribiremos las siguientes líneas de código:

```
ui->ly1->addWidget(multiSlider);
ui->ly2->addWidget(gauge);
ui->ly3->addWidget(scrollwheel);
```

Para vincular los dispositivos de control con los widget's.

- Deposito:

connect(ui->slider1,SIGNAL(valueChanged(int)),multiSlider,SLOT(setValue(int)));

- Tacómetro:

connect(ui->dial2,SIGNAL(valueChanged(int)),gauge,SLOT(setValue(int)));

- Válvula: Para el caso de la válvula, utilizamos la propiedad de pulsado del botón que hemos colocado. Esto se consigue haciendo click derecho encima del boto, seleccionamos *Go to slot* y elegimos *pressed()*.

```
void MainWindow::on_button3_pressed()
{
    scrollWheel2->changeValue(1);    //Cambia el estado
 }
```

En la Figura 7 vemos como quedaría el código dentro del archivo *mainwindow.cpp*.

mainwindow.cpp — MainWindow::MainWindow(QWidget *)

```
#include "mainwindow.h"
#include "ui_mainwindow.h"


#include <QtSvgDialGauge>
#include <QtMultiSlider>
#include <QtScrollWheel>


MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //DEPOSITO
    multiSlider = new QtMultiSlider(this);
    multiSlider ->setSkin("Deposito");       //Nombre de la carpeta con las imagenes
    multiSlider ->setValue(0);               //Valor inicial
    ui->ly1->addWidget(multiSlider);         //Coloca multiSlider en ly1
    connect(ui->slider1,SIGNAL(valueChanged(int)),multiSlider,SLOT(setValue(int)));

    //Tacometro
    gauge = new QtSvgDialGauge(this);
    gauge ->setSkin("Tacometro");          //Nombre de la carpeta con las imagenes
    gauge ->setNeedleOrigin(0.486, 0.466);
    gauge ->setMinimum(0);
    gauge ->setMaximum(360);
    gauge ->setStartAngle(-130);            //Angulo Minimo
    gauge ->setEndAngle(133);               //Angulo Maximo
    gauge ->setValue(0);                    //Valor inicial
    gauge ->setMaximumSize(200,200);
    ui->ly2->addWidget(gauge);              //Coloca gauge en ly2
    connect(ui->dial2,SIGNAL(valueChanged(int)),gauge,SLOT(setValue(int)));

    //Valvula
    scrollWheel = new QtScrollWheel(this);
    scrollWheel ->setSkin("Valvula");       //Nombre de la carpeta con las imagenes
    ui->ly3->addWidget(scrollWheel);        //Coloca scrollWheel en ly3

}

void MainWindow::on_button3_pressed()
{
    scrollWheel ->changeValue(1);       //Cambia de estado cada vez que pulsamos
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::changeEvent(QEvent *e)
{
    QMainWindow::changeEvent(e);
    switch (e->type()) {
    case QEvent::LanguageChange:
        ui->retranslateUi(this);
        break;
```

onstrucción | 2 Resultados de búsqueda | 3 Salida de la aplicación | 4 Salida del compilador

Figura 7

Haciendo lo anteriormente indicado, obtendríamos un programa con tres dispositivos para el control de los tres widget's que hemos insertado, como se observa en la Figura 8.

Figura 8

Ahora ya podríamos crear todos los widget's que deseemos sin mayor complicación, y dotándolos de movimiento a través de bucles, de este modo pueden estar interconectados todos los widget's entre si.